Surface reconstruction by computing restricted Voronoi cells in parallel

Dobrina Boltcheva, Bruno Levy

Surface reconstruction by computing restricted Voronoi cells in parallel

Dobrina Boltcheva[a,b,*], Bruno Lévy[b,a]

[a] *University of Lorraine, LORIA, CNRS, UMR 7503, Vandoeuvre-lès-Nancy, F-54506, France*
[b] *Inria, Villers-lès-Nancy, F-54600, France*

Abstract

We present a method for reconstructing a 3D surface triangulation from an input point set. The main component of the method is an algorithm that computes the restricted Voronoi diagram. In our specific case, it corresponds to the intersection between the 3D Voronoi diagram of the input points and a set of disks centered at the points and orthogonal to the estimated normal directions. The method does not require coherent normal orientations (just directions). Our algorithm is based on a property of the restricted Voronoi cells that leads to an embarrassingly parallel implementation. We experimented our algorithm with scanned point sets with up to 100 million vertices that were processed within few minutes on a standard computer. The complete implementation is provided.

Keywords: surface reconstruction, point cloud, restricted Voronoi diagram

1. Introduction

This article deals with surface reconstruction, a problem that may be formulated as follows: given a set of points sampled from a surface, recover the original surface from which those points were sampled. This general problem is motivated by numerous applications in reverse-engineering, prototyping, visualization, or computer vision since a growing variety of scanning devices nowadays provides measurements of objects in the form of point sets.

In practice, the common surface reconstruction pipeline usually involves several steps: After the acquisition with a laser scanner, for instance, the scans from the different views are first aligned into a common global coordinate system. Then they are filtered to reduce the input noise and remove some outliers. Then the surface normals at the points are estimated (and possibly oriented) and finally, a surface reconstruction algorithm builds a triangular mesh approximating the underlying surface.

Contribution: We propose a reconstruction method that is intended to be a part of the global reconstruction pipeline. It is designed to take as input an already filtered point set and connect the input points with triangles by computing their restricted Voronoi diagram.

The algorithm is similar in spirit to Localized Cocone [1] and Tangential Delaunay Complex [2]. As in these *local* algorithms, it does not compute any 3D data structure like a grid or a triangulation of the ambient space. The main difference is that our algorithm computes the intersection between the 3D Voronoi diagram of the points and a set of *disks* centered at the points and orthogonal to the estimated normal directions. We perform an iterative disk clipping in parallel for every point. As a consequence, our algorithm is both embarrassing parallel and memory efficient. It can handle large point sets, with up to 100 million points, in few minutes on an off-the-shelf PC.

Moreover, our algorithm dose not require consistent normal orientations, like in the Poisson methods [3], nor a scale value for every input sample, like in the Floating Scale Surface method [4]. It solely relies on estimated *normal directions*.

In addition, we also propose a manifold extraction algorithm and a set of post-processing heuristics to recover a clean mesh, specifically a 2-pseudo-manifold with or without boundaries [5].

Main limitations: Although our reconstruction algorithm is designed for already filtered point sets (where the points are assumed to be located exactly on the underlying surface), it is also useful in practice for real datasets (produced as unions of range maps where the points create some sort of "solid/thick surface"). For such datasets, the filtering becomes a mandatory pre-processing step which is a difficult issue and is beyond our current scope. Here, we use one or two iterations of a standard smoothing approach (projection onto the average plane of neighbors) to smooth the raw datasets which is often sufficient as confirmed by our experiments, see Figures 13, 12, 19 and 20. Note however that this smoothing is very moderate and does not wash out the small details, as shown on Fig. 16.

2. Related work

There are two classes of approaches which differ as to whether they approximate or interpolate the input points.

*Corresponding author
Email address: dobrina.botlcheva@univ-lorraine.fr (Dobrina Boltcheva)

Approximation methods. They aim at fitting a smooth surface to the input point set. These methods are well-equipped to handle various imperfections in the data such as noise, occlusion and registration errors. A recent state-of-the-art and a benchmark can be found in [6, 7]. Many of these algorithms [8, 9] compute a distance field by estimating the tangent plane at every point and computing closest distances using these tangent planes. The method of VRIP [10] takes advantage of the range scans acquired through laser triangulation to construct a volumetric signed distance field which merges the scans in a least-squares sense. The other approaches range from computing an indicator function [11, 3, 12, 13, 4], to locally fitting functions and moving least-squares methods [14, 15, 16]. In particular, the Poisson surface reconstruction method [11] solves for an approximate indicator function of the inferred solid, whose gradient best matches the input normals. The resulting scalar function is represented in an adaptive octree and is iso-contoured using a variant of the marching cubes algorithm [17]. Later, a Screened Poisson algorithm [3] has been introduced which resolves the over-smoothing problem and offers the ability to reconstruct meshes with boundaries. Recently, a Floating Scale Surface reconstruction method [4] has been proposed for multi-video-stereo applications that operates on large, redundant and noisy point sets but relies on a scale value for every input point which is estimated from the depth maps. Most of these methods require point sets with *consistently oriented* normals to estimated the signed-distance function. However, normal estimation in the presence of imperfect data remains an open problem [18, 19].

Interpolation methods. They elaborate upon Voronoi-Delaunay concepts and produce an interpolating surface in the form of a triangulation which uses a subset of the input points as vertices [20]. Often these algorithms extract this triangulation from the Delaunay triangulation of the input point set. The theory of ϵ-sampling [21] provides a solid background to study the properties of these algorithms. Extensive research effort has built on this theory, producing several *Restricted Delaunay based methods* such as Cocone [22], Super Cocone [1] and Power Crust [23] algorithms. Many other extensions have been compiled in a survey [24] and a monograph [25]. Among the Delaunay-based methods, there are the greedy approaches ([26], [27]), the Natural Neighbors ([9]), the "convection" algorithm [28], the sampling theorems introduced in [29], and the WRAP algorithm together with its analysis in [30, 31].

To avoid the high computational cost associated with computing the 3D Delaunay triangulation of the point set, several *local* computation methods were proposed. For example, *The Ball Pivoting algorithms* (BPA) [32, 33] are local, optimized variants of the Alpha Shapes [34]. These algorithms can be parallelized to handle large datasets [35]. The method introduced by Digne et al. [36] improves the noise-resilience of the BPA by applying it to a scale-space version of the input point set.

Some *local* variants of the Cocone algorithm were also developed to mitigate the scalability problem. These algorithms are local in the sense that no 3D data structure like a grid or a triangulation of the ambient space is computed and the final triangulation is obtained by gluing local triangulations around each sample point. Funke and Ramos [37] demonstrated that the Cocone algorithm can be modified so that no global computation of the 3D Delaunay triangulation of the entire point set is required. They showed that with a *locally uniform* ϵ-sampling the Cocone of each point is a local object and it is completely determined by nearby samples. Dumitriu et al. [38] used this framework to design a surface reconstruction algorithm with correctness guarantees. Although the algorithm is quite complicated, they have produced an implementation [39]. Amenta and Kil [40] employed similar techniques and designed a parallel algorithm operating on the GPU.

Alternatively, Dey et al. [1] developed an octree-based version of the Cocone algorithm such that the 3D Delaunay triangulation is only computed on small clusters of the initial point set. By adopting the assumption that the local sampling density is bounded by a constant [41], they showed that the theoretical guarantees of the original Cocone algorithm are maintained. However, when the input point set does not meet the sampling requirements, it is necessary to use some heuristics in the manifold extraction step.

Another promising idea to deal with the scalability problem is to restrict the computations to planes [42, 43] or to the tangent space [2]. The tangential complex is obtained by gluing local (Delaunay) triangulations around each sample point and the neighborhoods are "reconciled" with a "Star-Splaying" approach [44]. Although, this has been the first theoretical algorithm able to reconstruct a smooth closed manifold in a time depending only linearly on the dimension of the ambient space, to the best of our knowledge, there is no existing implementation.

3. Computing the restricted Voronoi diagram in parallel

The general work-flow of our algorithm is summarized in Figure 1. The input is a set of 3D points, with normals or not. Note that no consistent orientation of the normals is required but only directions. The output of the algorithm is a triangular mesh connecting the input points which approximates the surface. Specifically, the output triangulation is a compact 2-pseudo-manifold (see Section 3.5), it is orientable (no Moebius strip) and may contain several connected components and boundaries.

The next sections describe each step in detail, as well as the parameters they depend on. The first three steps, kd-tree construction §3.1, point set smoothing §3.2 and normals estimation §3.3 use classical methods.

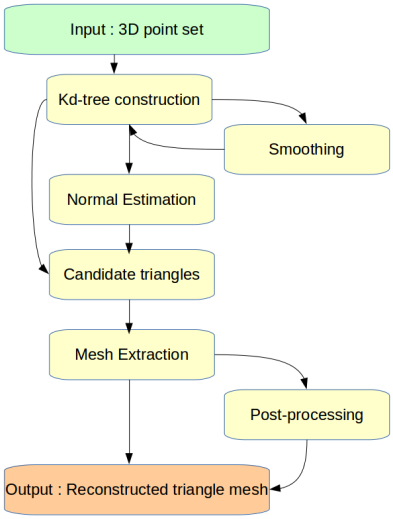


Figure 1: Overview of the algorithm.

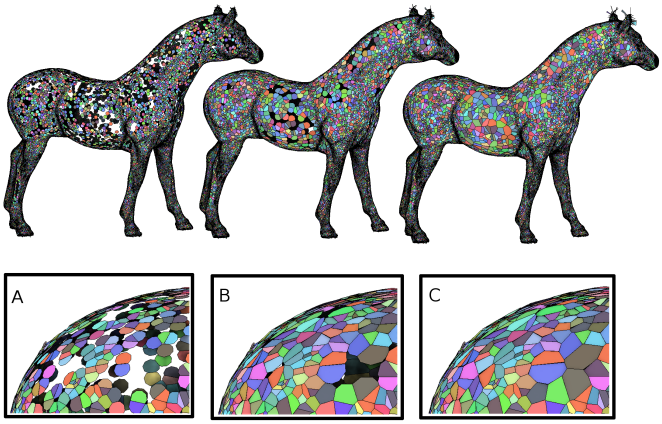


Figure 2: We compute the Voronoi diagram of the input point set restricted to a union of disks centered on the points and orthogonal to the estimated normals. From A to C: the radius of the disks is increased from 0.2% to 4% of the bounding box diagonal. All the restricted Voronoi cells (colored polygons) are computed in parallel.

Our main contribution is the *candidate triangles* step (§3.4), that computes the intersection between the Voronoi cells of the input points with a set of disks centered on the input points. It shares some similarities with the theoretical method of Funke and Ramos [37], the Localized Cocone algorithm [1], the Tangential Delaunay Complex [2] and the Star Splaying [44]. Instead of computing the Delaunay triangulation *then* restricting it, we *directly* compute the restricted Voronoi cells (intersections between the Voronoi cells and the disks), and deduce the restricted Delaunay triangles from the combinatorial information of the restricted Voronoi vertices (see Section 3.4, Figure 2, Figure 4 below). This results in an algorithm which is simple and easily parallelizable.

We also introduce a new *manifold mesh extraction* method (§3.5) which avoids the trimming step used in previous work (the most critical because it can end up with an empty mesh) [22]. Starting from a subset of candidate triangles which forms a clean and orientable 2-manifold with boundaries, we add the remaining triangles, one by one, in an arbitrary order, only if they do not break the topological properties of the initial mesh. The algorithm terminates with the largest possible non-empty manifold mesh.

3.1. Kd-tree construction

The basic geometric operation in our algorithm is finding the nearest neighbors of a point. To optimize this operation, we organize the input points in a kd-tree [45]. To do this, one may use the *Approximate Nearest Neighbors (ANN)* library [46] which, with ϵ set to 0, computes the exact nearest neighbors for every point. But the memory consumption can be significantly reduced by replacing the kd-tree used in ANN with a *balanced* binary tree, with links that are completely implicit. This makes it possible to process large point sets (up to 50 million points) on a desktop PC with 16 Gb RAM. Our implementation is given in the supplemental material.

3.2. Smoothing

Since our algorithm interpolates the input points, it requires a clean and accurately registered point set. Thus, if the input set is noisy such as raw scanning data, it has to be smoothed before the reconstruction. Note however that any point cloud smoothing technique can be used at this step.

Our current implementation uses the projection onto the best approximating plane of the k-nearest neighbors, [8]. Specifically, for a point p and its k nearest neighbors $\{p_i\}_{i=1}^{k}$, we find the fitting plane $n^T x = c$ for p by minimizing the error term $e(n,c) = \sum_{i=1}^{k}(n^T p_i - c)^2$ under the constraint $n^T n = 1$. Then, the point p is projected onto this plane. We compute the best fitting plane for each point in parallel. Once all the points are projected, we update the kd-tree.

Parameters. This step uses two parameters: the number of nearest neighbors *nb_neighbors* and the number of smoothing iterations *smooth_iter*. We used 30 neighbors for all datasets, and 1 or 2 smoothing iterations for the raw scanned data.

3.3. Normal estimation

If the input points come without normals, our algorithm estimates them locally with the best fitting plane as in the smoothing step. Note however that the algorithm does not need to *orient* the normals, it only needs a normal *direction* at every point which is used to create a disk in the subsequent step of the algorithm.

3.4. Candidate Triangles

Given a point set $\{p_i\}_{i=1}^n$ with estimated normals $\{n_i\}_{i=1}^n$ and a radius r, our algorithm computes the intersection between the Voronoi cells of the points $V(p_i)$ and the disks D_i^r of radius r centered on the points and orthogonal to the normals.

During this step, we fist build a disk D_i^r at every point p_i in the input point set P orthogonal to n_i and with user-given radius r. In our implementation, we use a polygon (with 10 vertices) that approximates the disk. In contrast with previous algorithms, we *explicitly* build the *restriction* of the Voronoi cell $V(p_i)$ to the disk D_i^r at p_i, $V(p_i) \cap D_i^r$, as shown on Fig. 2 and Fig. 4. Thus we collect a set of *restricted* Voronoi vertices. The set T of *candidate triangles* is then defined by the triangles which are dual to the restricted Voronoi vertices (more on this below).

Note that the disk radius has to be chosen in function of the sampling rate and it entails a trade-off between accuracy and computation speed. A very small radius leads to uncovered holes in the final mesh (2(A) and (B)), while a very large one increases the computation time (since the algorithm looks for neighbors in a larger space).

The basic operation in this step of the algorithm consists of clipping the disk D_i^r with the Voronoi cell $V(p_i)$ of every point p_i. Here we use ideas that are similar to those presented in the Localized Cocone algorithm [1] which allows us to build locally the candidates triangles by using only the k-nearest neighbors. More precisely, we use the clipping algorithm and the local characterization of the restricted Voronoi cells introduced in [47] which we recall in the following in order to make the explanation self-contained.

Computing Voronoi cells through iterative clipping

Let us recall that the Voronoi cell $V(p_i)$ corresponds to the intersection of the halfspaces : $V(p_i) = \bigcap_{j \neq i} \Pi^+(i,j)$, where $\Pi^+(i,j)$ denotes the halfspace bounded by the bisector of (p_i, p_j) that contains p_i. Note that the bisectors between p_i and all the other points are involved in the definition above, whereas only a small subset of them corresponds to actual Voronoi edges (see Figure 3). We can classify the bisectors $\Pi^+(i,j)$ into two sets: *contributing* and *non-contributing*. Clipping a Voronoi cell by a non-contributing bisector does not change the result. Therefore, the intersection of any superset of the contributing bisectors corresponds to the Voronoi cell. Let $p_{j_1}, p_{j_2}, \dots p_{j_{n-1}}$ denote the vertices sorted by increasing distance from p_i. Let V_k denote the intersection of the k first halfspaces and let R_k denote its p_i-centered radius:

$$\begin{aligned} V_k(p_i) &= \bigcap_{l=1}^{k} \Pi^+(i, j_l) \\ R_k &= \max\{d(p_i, x) | x \in V_k(p_i)\}. \end{aligned}$$

Data: the index i of the point x_i, the disk D_r centered on it, the number of neighbors n and the set of points P
Result: Restricted Voronoi Facet at p_i: $RV(p_i) = V(p_i) \cap D_i^r$

$RV \leftarrow D_i^r$
$R_k \leftarrow \max\{d(p, p_i) | p \in RV\ \}$
$k \leftarrow 1$
while $d(x_i, x_{j_k}) < 2R_k$ *f and* $k < n$ **do**
  $RV \leftarrow RV \cap \Pi^+(i, j_k)$
  $R_k \leftarrow \max\{d(p, p_i) | p \in RV\}$
  $k \leftarrow k + 1$
end

Algorithm 1: *Computes a restricted Voronoi cell as the intersection between a tangential disk and the Voronoi cell of a point.*

The configuration in Figure 3 suggests that for all j such that $d(p_i, p_j) > 2R_k$, the bisector $\Pi^+(i,j)$ is non-contributing, i.e. $V_k(p_i) \subset \Pi^+(i,j)$. This observation can be formally proven as follows: Consider $p \in V_k(p_i)$ and p_j such that $d(p_i, p_j) > 2R_k$. By definition of R_k, $d(p, p_i) < R_k$. We have $d(p_i, p) + d(p, p_j) \geq (p_i, p_j)$ (triangular inequality) and $d(p_i, p_j) > 2R_k$, therefore $d(p, p_j) > R_k > d(p, p_i)$ and $p \in \Pi^+(i,j)$.

As a direct consequence:

$$d(p_i, p_{j_{k+1}}) > 2R_k \Rightarrow V_k = V(p_i).$$

We call *radius of security* the first value of R_k that satisfies this condition. Note however that this observation does not have a practical value in the case of the unrestricted Voronoi diagram, since some cells are unbounded and have infinite R_k (for an infinite cell, all the bisectors are considered, leading to prohibitive computation time). However, the observation can be clearly used to compute the restricted Voronoi cells $RV(p_i) = V(p_i) \cap D_i^r, \forall p_i \in P$, since they are finite (thay are contained in D_i^r).

We use Algorithm 1 to build the restricted Voronoi cell for every point, which is the intersection between the disk D_i^r, centered on it and orthogonal to the estimated normal and the Voronoi cell of the point.

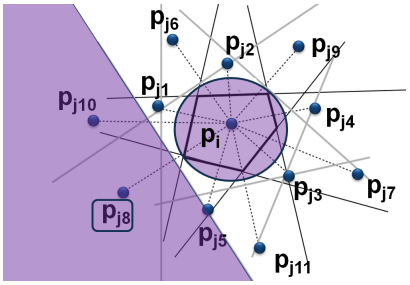


Figure 3: The Voronoi cell $V(p_i)$ is the intersection of the halfspaces defined by the bisectors of the segments $[p_i, p_j]$. Once a point (here p_{j_8}) is further away twice than the current Voronoi cell at p_i, it no longer contributes to the cell.

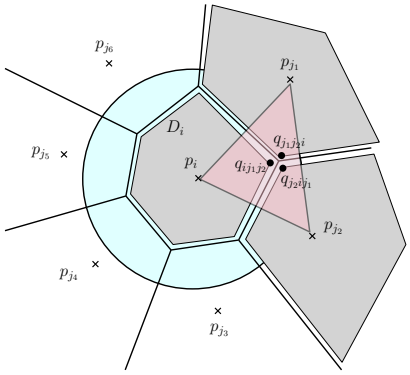


Figure 4: The Delaunay triangles (p_i, p_{j_1}, p_{j_2}) are deduced from the indices of the bisectors that generate the Voronoi vertices q.

Note that every restricted Voronoi vertex q is the intersection between the disk D_i^r and the bisectors Π_{i,j_1} and Π_{i,j_2}. We deduce from the indices j_1 and j_2 the Delaunay triangle i, j_1, j_2 associated with each restricted Voronoi vertex, see Fig. 4. We denote by T the so-defined set of *candidate triangles*.

Parameters. This step uses one parameter: the disk *radius* (in % of the bounding box diagonal). Note that when the radius r is larger than the width of the Voronoi cells, the disks D^r touch all the sides of their Voronoi cells (like on Figure 2 (C)), and increasing the radius no longer changes the result. Thus, in our experimental results, we set the radius to a large value, r=5% of the bounding box diagonal. For very large datasets (> 10M vertices), since Voronoi cells are smaller, we decrease the size (r=0.5% bbox diagonal) to make computations faster because the algorithm looks for the neighbors within a smaller space.

Structure of the set of candidate triangles

For applications that just need to visualize the surface, providing the set of candidate triangles may be sufficient. However, most of the applications in Geometry Processing require clean manifold meshes with coherent orientations for further processing. This is why we present hereafter a manifold extraction method which uses a set of heuristics to build a valid mesh from the soup of triangles T.

We first need to take a closer look on the structure of T. The set T of candidate triangles is composed of the restricted Delaunay triangles T_3 plus some other triangles, as explained below.

The set T_3 of restricted Delaunay triangles is defined as:

$$\begin{aligned} T_3 = \{(i,j,k) \mid & (D_i^r \cap V(p_j) \cap V(p_k) \neq \emptyset) \text{and} \\ & (D_j^r \cap V(p_i) \cap V(p_k) \neq \emptyset) \text{and} \\ & (D_k^r \cap V(p_i) \cap V(p_j) \neq \emptyset)\}. \end{aligned} \quad (1)$$

In other words, they correspond to triples of restricted Voronoi cells $(RV(p_i), RV(p_j), RV(p_k)$ that are mutually in contact (see Figures 2 and 4). These triangles can be easily found, by generating all the index triples that correspond to the restricted Voronoi vertices, sorting them (using, for instance, the lexicographic order) and keeping the triples that appear three times in the sorted sequence. The so-computed T_3 triangles are displayed in Figure 5-A. Note that in the sorted sequence, there are also triples that appear once or twice. The corresponding set of triangles is referred to as $T_{1,2}$ and is defined by:

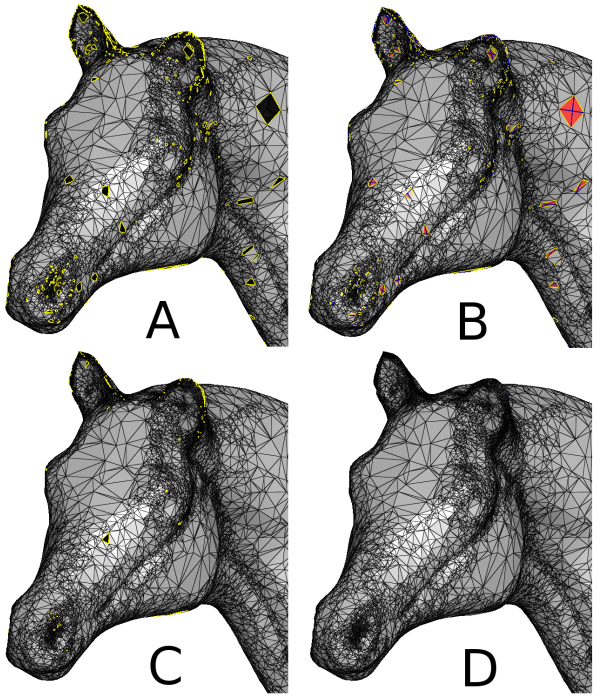


Figure 5: A: the set of triangles seen from three restricted Voronoi cells (T_3), with some holes (yellow). B: the set of triangles seen from one or two Voronoi cells ($T_{1,2}$, in red) fills some holes, but have non-manifold configurations (four red triangles in the right). C: result after carefully inserting the $T_{1,2}$ triangles one by one. D: suppressing the remaining holes.

$$T_{12} = \{(i,j,k) \mid D_i^r \cap V(p_j) \cap V(p_k) \neq \emptyset\} - T_3. \quad (2)$$

This means that a triangle (i,j,k) is in T_{12} whenever the restricted Voronoi cell of p_i "sees" p_j and p_k but not both conversely. Some configurations are shown in Figure 6 that correspond to nearly co-spherical points. Since the restricted Voronoi vertices are computed independantelly for every sample point, the numerical computations lead to vertices whose cooridantes differ slightly du to the finite computer precision. Therfore, there are restricted Voronoi vertices with different cardianlities - the majority are computed for exactly three neighoring samples, but there are also restricted vertices obtained only from 1 or 2 sample points.

The so-defined T_{12} triangles are displayed in red in Figure 5-B. As expected, they are not as "reliable" as the T_3 triangles, in the sense that they are likely to generate non-manifold configurations (see for instance the four interconnected red triangles that form a "sliver" in the top right part of the neck in Figure 5-B). However, one can observe that they contain interesting information, and could be used to fill the holes in the T_3 triangles (Figure 5-A).

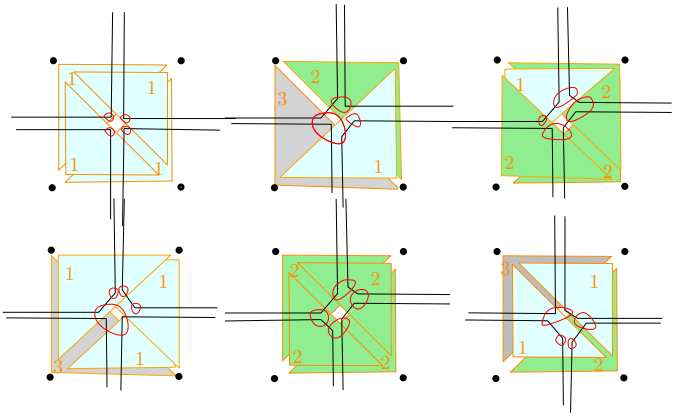


Figure 6: Configurations of 4 co-circular sample points (black dots) with their partial restricted Voronoi facets (sketched with black segments). The dual T_{12} triangles are shown with different colors corresponding to their cardianlities.

Thus, the idea is to start from the T_3 triangles, then carefully insert the triangles from the T_{12} set, one triangle at a time, making sure that the updated mesh remains manifold. If it is not the case, the triangle is rejected. The associated algorithm is detailed in the next section.

3.5. Manifold mesh extraction

We first build a clean orientable 2-manifold mesh with a subset of the T_3 triangles. Then we fill the holes iteratively, with the remaining triangles, if they do not jeopardize the topological properties of the output.

The mesh structure that we build incrementally is a manifold mesh where every edge can only be adjacent to two facets, edges with only one incident triangle are allowed, as well as triangles adjacent only by one vertex (this is a non-manifold vertex whose neighborhood is not a topological 2-sphere). The output mesh (also known as a 2-pseudo-manifold [5]) is orientable and may have boundaries and several connected components.

We initialize the output mesh with all T_3 triangles. Then we remove all the triangles incident to non-manifold edges and *non-manifold vertices by excess*. Recall that a non-manifold edge is incident to more than 2 triangles, while a 'by-excess' non-manifold vertex has a closed loop of triangles (clean umbrella) in its neighborhood plus additional triangles. At this point, we also orient the initial mesh and check that there are no Moebius strips within it. If the algorithm finds a Moebius strip, it removes the last triangle that closes the loop. Note that the initial mesh can however exhibit triangles incident to each other only by a vertex.

We then fill the holes of the initial mesh with the remaining triangles from T_{12}. The T_{12} triangles are tested against the following criteria, from the simplest to the most time consuming. Only the triangles that pass all the tests are inserted in the final surface.

- **Geometric test**: ensures that the normals of the neighboring triangles agree, meaning that they do not make a sharp angle (less than $\pi/3$).
- **Combinatorial test I**: ensures that the triangle is properly connected to the current mesh. Every new triangle should be either incident to two edges of existing triangles, or to one edge of an existing triangles and one isolated point.
- **Combinatorial test II**: checks for non-manifold edges and tests whether the three candidate edges are manifold.
- **Combinatorial test III**: checks that inserting the new triangle do not generate 'by-excess' non-manifold vertices.
- **Combinatorial test IV**: checks if in the neighborhood of the triangle the same connected component appears with two opposite orientations, then connecting the triangle would create a Moebius strip. The triangle is rejected if it is incident to the same connected component with two different orientations.

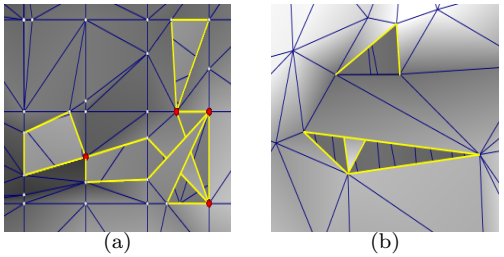
The output of the algorithm is the largest possible triangular mesh that connects the input points. It is orientable 2-manifold (without non-manifold edges) but it can have non-manifold vertices, such as a pinched torus. It possibly has several connected components and boundaries.

Parameters. This step uses one parameter which is the maximum deviation angle between the triangle normals *max_N_angle*. We always use the default value set to $\pi/3$.

3.6. Post-processing

At this point, after the manifold extraction step, the resulting mesh may still contain a number of isolated holes, caused mainly by insufficiently sampled sharp features (see e.g. the top of the ears of the horse in Figure 11 C). To eliminate these, we add an extra post-processing step, that fills some of the holes.

The algorithm takes an user-defined *maximum hole size* parameter, in terms of number of edges. Then it looks for *simply connected holes* and fills them using a classical



(a) (b)

Figure 7: (a) Simple bridge. (b) Double bridge.

Model	Size #M v.	Ours sec.	SCocone sec.	ScSpace sec.	ScrPois. sec.	Wavelets sec.
hotdog	0.001	0.07	0.22	11	2	10
sphere	0.03	0.24	2	2	8	85
horse	0.1	1	13	KO (3004)	19	KO (50)
daratech	0.24	2	35	37	28	103
anchor	0.26	2	35	crash	49	174
dame	0.31	3	45	40	46	120
hand	0.32	3	52	KO (922)	59	KO (80)
lord quas	0.35	4	54	46	32	108
bunny	0.36	5	50	29	49	201
dc	0.46	4	71	32	49	179
MagaliHand	0.7	8	109	KO (591)	48	83
eagle	0.79	11	113	127	118	177
Nasa	0.88	10	117	105	105	232
Cube	1.01	14	139	162	134	268
PerfumeBottle	1.25	13	172	188	122	252
Galaad	1.45	17	210	369	80	159
ToyTurtle	1.47	16	207	303	100	179
SophieHand	1.58	19	217	334	107	197
chineseDragon	1.76	25	250	KO (899)	149	273
PooranHand	2.10	22	313	480	94	197
PierreHand	2.63	27	380	643	116	248
TrungHand	2.83	42	409	699	132	275
HeleneHand	3.65	39	544	736	119	297
cutanubis	5	45	crash	KO (2522)	84	321
anubis	9.99	88	1807	2153	170	632
night	11.05	110	1806	crash	567	727
lucy	14.02	132	2308	1483	267	911
tanagra	16.38	173	2673	2302	274	1040
facadeValGrace	29.46	341	KO (4730)	crash	856	1916
chateauRives	32.75	361	crash	crash	KO (543)	2050
rosetta	36.2	352	crash	crash	676	2228

Figure 8: Comparison of processing times for the methods. Timings are in seconds and model sizes in million vertices. We used implementations written by the authors of the cited methods, and the same computer for all the tests. For Poisson and Wavelets we used an octree with depth 10. ScaleSpace and SuperCocone adapt the depth of their octrees (from 6 to 10). KO means that the result has too many holes or is mostly unrelated with the data or the computation time explodes as compared to data of similar size.

loop-split algorithm [48]. This algorithm splits the hole recursively until it gets holes composed of exactly 3 edges and fills them with the corresponding triangle.

In order to ensure that the holes are simple loops, we first search and remove the configurations where the holes contain "bridges"(see Fig.7). We call bridge triangles those that are visited more than once when we walk around the border edges of a hole. Note that this definition dose not capture bridges wider than two triangles but we did not encounter any, in practice.

During this last post-processing step, we can also remove some remaining small components, in function of their size (in number of facets).

Parameters. This step uses the following two parameters which give the size of the holes to be filled and the connected components to be removed. In our experiments, we have always used:

- *max_hole_edges* (=500) : Fill holes with a smaller number of edges
- *min_comp_facets* (=10) : Remove small components (in facet number)

4. Experimental results and Discussion

This section details our empirical results and gives comparisons with four state-of-the-art approaches implemented by their authors (ScaleSpace reconstruction [51], Screened Poisson [3], SuperCocone [1] and Wavelet reconstruction [12]). We evaluated the methods both in an artificial scenario with point sets sampled from a known reference surface (§4.1) and with a database of models with sizes ranging from a few thousand to 100 million points (§4.2).

4.1. Comparison - Hausdorff distance to reference surface

We first evaluate experimentally whether our method reconstructs the same surface as previous work, under different sampling conditions. We created a point set by sampling a given reference surface and measured the Hausdorff distance between the reference surface and the output of the reconstruction methods. The samplings are shown in Figure 9. We tested our method both without and with post-processing (holes filling). The results are reported in Table 1. Our method gives an output with a deviation from the reference surface similar to the state-of-the-art methods.

4.2. Real-scale data

We experimented our algorithm with datasets from public repositories [52, 53, 54, 55, 49, 50] and the data used in the original articles of the four state-of-the-art methods we compared with. We conducted different experiments, to understand the behavior of our method and the previous work with point sets of different sizes (from a few thousands to 100 million points) and with point sets with different structures (non-uniform sampling, small details, noise, overlapping scans).

The table shown on Figure 8 lists some of the models and reports the timings for the algorithms. We used implementations written by the authors of the cited methods, and the same computer for all the tests [Processor: Intel Core i7-4900MQ CPU @ 2.80GHz, RAM:16Gb]. We measured the timings by using the TIME function in Linux. Thus, they indicate all computation steps (loading and saving included) for all methods. In particular, they comprise the pre- and post-processing steps of our method. Size-time

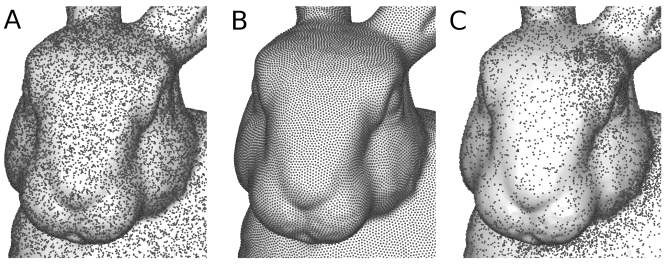


Figure 9: The used sampling patterns. A: homogeneous; B: homogeneous regularized; C: curvature adaptive

	homo_100K	homo_1M	homo_reg_100K	homo_reg_1M	adapt_100K	adapt_1M
ours : $M1 \rightarrow M2$	0.58	0.18	0.58	0.51	0.56	0.4
ours : $M2 \rightarrow M1$	0.03	0.0002	0.08	0.03	0.015	0.008
ours-no post-proc: $M1 \rightarrow M2$	0.5	0.17	0.58	0.31	0.35	0.18
ours-no post-proc: $M2 \rightarrow M1$	0.027	0.0002	0.07	0.03	0.015	0.008
Screened Poisson : $M1 \rightarrow M2$	0.6	0.35	0.6	0.56	0.58	0.40
Screened Poisson : $M2 \rightarrow M1$	0.17	0.23	0.09	0.1	0.28	0.16
Scale Space : $M1 \rightarrow M2$	0.8	0.3	0.9	0.51	4.2	0.7
Scale Space : $M2 \rightarrow M1$	0.027	0.0002	0.07	0.03	0.01	0.008
Super Cocone : $M1 \rightarrow M2$	0.58	0.18	0.58	0.51	1.22	0.4
Super Cocone : $M2 \rightarrow M1$	0.027	0.0003	0.07	0.03	0.01	0.008
Wavelets : $M1 \rightarrow M2$	0.52	0.50	0.57	0.57	3.6	2.8
Wavelets : $M2 \rightarrow M1$	0.44	0.12	0.25	0.05	3.6	11

Table 1: For each algorithm, the table reports the two one-sided distances $d(M1, M2)$ and $d(M2, M1)$ between $M1$ (the reference mesh) and $M2$ (the mesh built by the algorithm). Distances are expressed as percentages of the diagonal of the mesh bounding box.

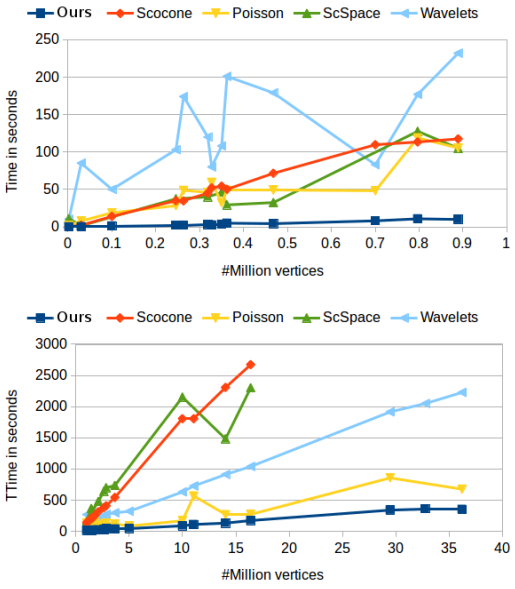


Figure 10: Performance of the tested algorithms. Up, timings obtained on the Small data. Bottom, timings obtained on the Large data.

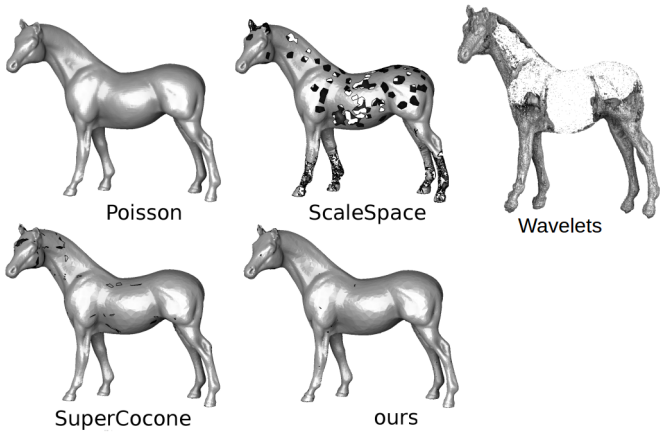


Figure 11: Comparison of the algorithms with a small dataset (100K vertices) with high variations of sampling density.

curves for all methods and datasets are shown in Figure 10.

To be able to analyze and compare qualitatively the behavior of the algorithm, we deactivated the post-processing step (hole filling). In the following (unless specified otherwise), we compare the output of our manifold extraction step with the outputs of the four methods cited before.

Non-uniform sampling

Figure 11 shows a horse sampled with 100K points with a varying sampling density. ScreenedPoisson reconstructs a perfect surface in 19 sec, while Wavelets fails to reconstruct this type of data. ScaleSpace has difficulties finding the correct scale in this dataset, and takes more than 3000 seconds. The result has holes when the triangles are larger than the estimated scale. SuperCocone divides the input points into several subsets using an octree. It sometimes fails to recover triangles that cross octree cells boundaries. Our method misses some triangles resulting into small holes that can be filled in with our post-processing step.

Small details

Figure 12 compares the results on a clean point set having many small bumpy features (the feathers of the eagle), [55]. Poisson faithfully reconstructs the features. Wavelets performs similar to Poisson. ScaleSpace reconstructs a continuous manifold mesh that also follows the bumpy features. SuperCocone and our algorithm give similar results, with more missing triangles than ScaleSpace. However, the missing triangles in our reconstruction are mostly small holes that can be efficiently removed by our post-processing step.

Noise

Real datasets such as raw scans produced as unions of range maps usually exhibit alignment errors. In this data, the points are not located exactly on the underlying surface and form some sort of "solid surface". As it has been stressed previously, our method requires a clean point set since it connects the input points with triangles. The method typically fails when registration errors create two layers of points in overlapping regions of different scans (see Fig. 15). However, managing the over-sampling and the noise of the dataset is an active research subject which is beyond our scope. To accommodate the raw scans in our database we have used 1 or 2 iterations of our simple pre-processing smoothing step which have worked in most cases, as shown by our experiments. Note that the

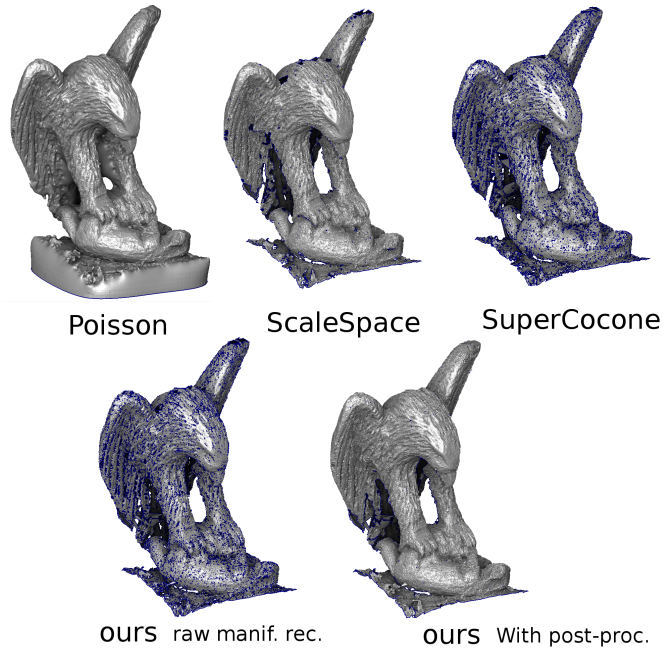


Figure 12: Comparison of the algorithms with a clean dataset with small-scale bumpy features.

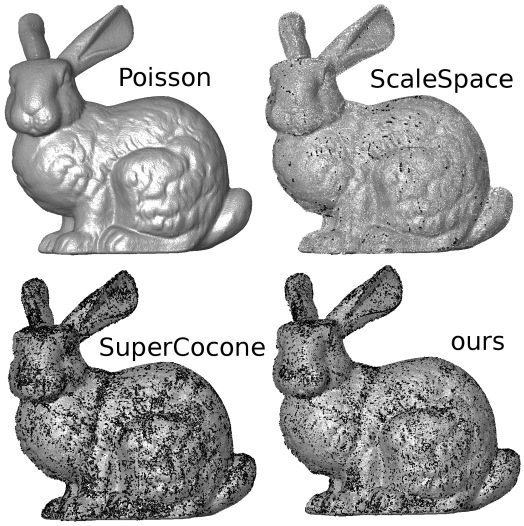


Figure 13: A noisy dataset (original scanner data of the Stanford Bunny). Poisson both filters out the noise and recovers the fine features. ScaleSpace successfully extracts a closed manifold mesh. SuperCocone and our algorithm recover a smaller number of triangles.

smoothing time is included in the timings given in Figure 8.

In Figure 13, we show the behavior of the algorithms for a noisy point set (original registered dataset for the Stanford bunny [53]). For this data, Poisson and Wavelets both cancel the noise and recover the fine details. ScaleSpace manages to extract a manifold mesh that exactly follows the details of the data, even when it is very bumpy. Our algorithm outputs a result that is very similar to that of SuperCocone. The same dataset with one iteration of pre-processing smoothing is shown in Figure 14. All algorithms had the same smoothed input and gave similar results. Notice the aligned missing triangles in SuperCocone at boundaries between adjacent octree cells.

We stress, however, that our method needs a very moderate smoothing which does not wash out the small details. With the Tanagra model (see Fig.16), we reconstruct the stamp with the same resolution as Screened Poisson and Scale Space, after 2 iterations of smoothing.

Impact of pre and post-processing steps

Figures 17 and 18 show raw LiDAR scans with respectively 14M and 12M input points (from [49]) and illustrate the impact of the pre-processing smoothing and the post-processing hole filling steps of our method.

The mesh in Fig. 17, built without pre-processing (in 200s), is quite noisy since it reconstructs the data noise with 123 connected components and 7250 border edges. The mesh generated with 1 iteration of pre-processing (in 123s) is smoother and has only 15 connected components and 1585 border edges.

Figures 18 demonstrates the same behavior. Our method manages to reconstruct the raw scans but the resulting meshes have many holes that are left to be filled in by the post-processing step. On this data the smoothing step takes 10s and the post-processing step - 30s. The raw candidate triangles generation is done in 30s while the manifold extraction takes between 20s, if the points are smoothed, and 100s otherwise. The total reconstruction times range from 90s to 190s.

Large datasets

For the large datasets, existing Delaunay-based methods (ScaleSpace and SuperCocone) start encountering memory limitations. ScaleSpace crashes on all examples. SuperCocone gives an incorrect result on one of them ('facade-val-de-grace') since many triangles are missing and crashes on the others ('chateau-rives' and 'rosetta'). The results are displayed in Figures 19 and Figure 20. A rendering of 'rosetta' is available in the supplemental material.

Results of the 'facade-val-de-grace' are shown in Figure 19. For this dataset, Poisson has problems with some incoherent normals, causing bubbles to appear. Wavelets have similar behavior. It may not be difficult to fix, but the need for having consistent normal orientations is an issue for using approximation based methods. SuperCocone missed many triangles, probably because the implementation starts to encounter memory problems.

We also show, in Figure 20, the 'chateau-rives' example with 33M points. For this dataset, our algorithm outputs a valid result with 65M facets in 361s. Among the other methods, only Wavelets build an approximating coarse mesh with 3M facets in 2050 seconds, with octree depth=10 (See Fig.20).

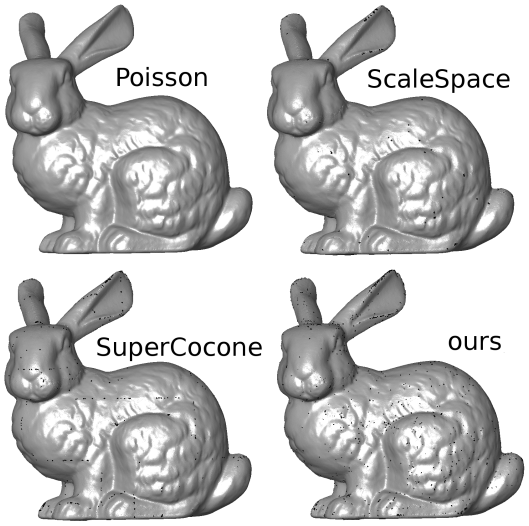


Figure 14: The same dataset as in Figure 13 with one iteration of smoothing applied. The methods give similar results.

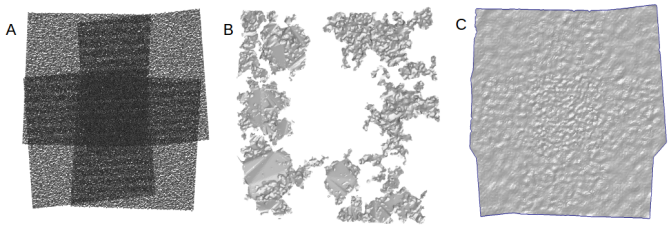


Figure 15: A: overlapping scans; B: our reconstruction without smoothing; C: our reconstruction with one iteration of smoothing (data from [4])

Figures 22 and 21 show reconstructions of raw scans from the "Large-Scale Point Cloud Classification Benchmark" repository [50], with respectively 14M and 10M of points. The market place mesh in Fig.22 has 20M of triangles that have been generated in 200s, with 1 iteration of smoothing (done in 15s). The saint Gallen mesh in Fig.21 has 28M of triangles generated in 280s, with 1 iteration of smoothing (done in 20s). These meshes are far from being perfect but can be used as initial reconstructions for some downstream geometry processing applications.

Finally, we tested our method with 100 million points (sampled randomly on the Stanford bunny) which does not fit in 16Gb of RAM. Using a 32Gb computer, our method generates 600M candidate triangles in 90s, and then takes 140s to output the raw manifold mesh. With post-processing, the complete algorithm takes 533s and the final mesh has 200M facets. Among the other methods that we tested, only Poisson and Wavelets can also process this dataset. With octree depth=10, Poisson takes 1531s to output a mesh with 3M vertices and 6M facets and Wavelets takes 387 seconds for a mesh with 10M vertices and 5M facets. With octree depth=12, Poisson takes 5809s to output a mesh with 48M vertices and 96M facets and Wavelets takes 762 seconds to build a mesh with 162M vertices and 81M facets. With depth=13, both do not fit in 32 Gb.

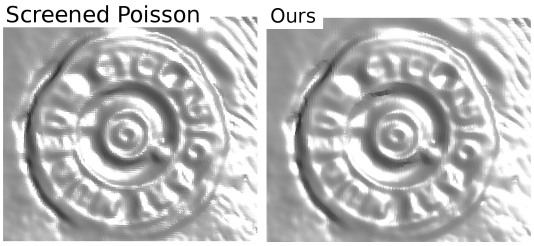


Figure 16: Reconstruction of the Tanagra sculpture (detail). Left: Screened Poisson reconstruction (octree depth=12). Right: our result (with 2 iterations of smoothing) where the fine-scale details are preserved.

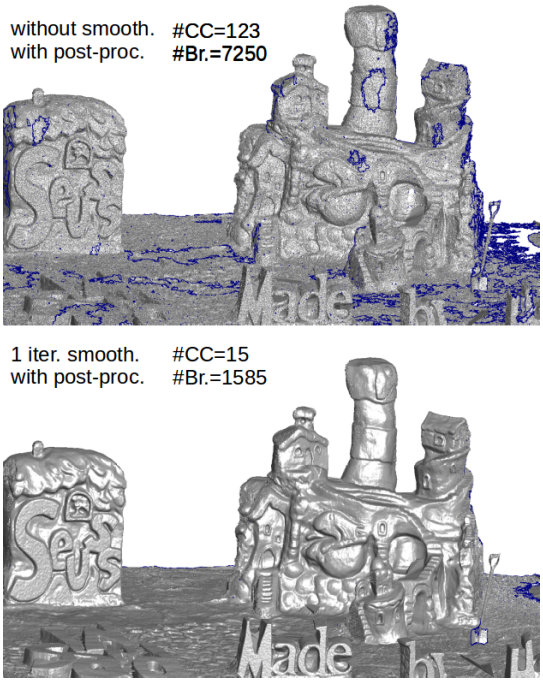


Figure 17: A raw LiDAR scan (14M points) of a sandcastle (note the shovel). Top, our mesh reconstructed with no pre-processing (built in 200s) Bottom, our result reconstructed after 1 iteration of smoothing (built in 123s).

5. Conclusion and Discussion

In this paper, we have presented a surface reconstruction algorithm from a set of points in 3D based on a simple observation about the restricted Voronoi cells. Our algorithm has a trivial implementation, is embarrassingly parallel and economic in terms of memory resources. Therefore it scales up very well to datasets with tens of millions vertices while keeping computation times smaller than 5 minutes. Its simplicity makes it possible to embed it into devices such as smart-phones and cameras with a CPU (an Android ARMV7 implementation is provided in the supplemental material).

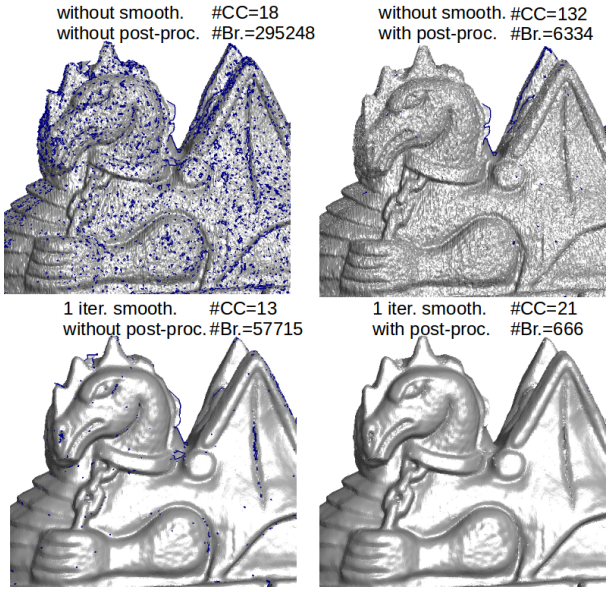


Figure 18: A raw LiDAR scan (12M points) from [49]. Ours results in function of pre- and post-processing steps.

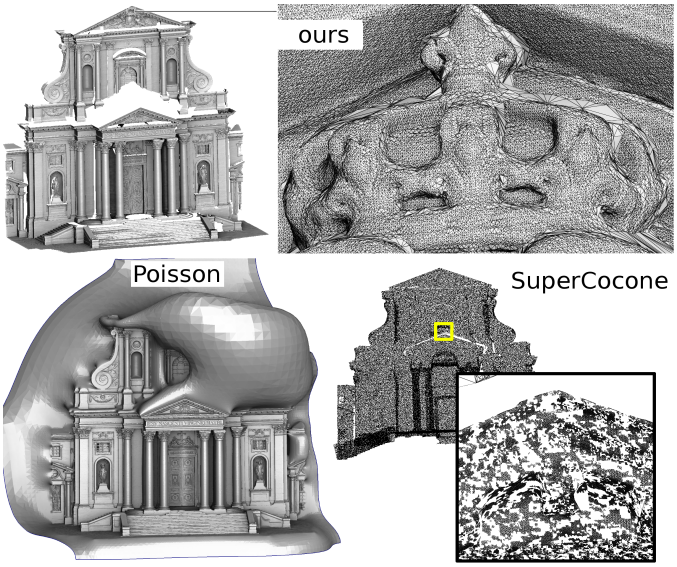


Figure 19: A raw scanned point set with 30M vertices. On this dataset, some normals are not correct causing some bubbles in the Poisson reconstruction (built in 856s). SuperCocone gives a result in 5011 seconds, but has many missing triangles (maybe it starts reaching memory limits). Our algorithm (with 2 iterations of smoothing) takes 340s

Compared with the state-of-the-art methods, it is not as good as Poisson and Wavelets for filling the holes, as expected since these two methods build smooth approximation meshes. Concerning the interpolation methods, our method has a behavior similar to the Cocone and ScaleSpace even if the latter may have a better set of candidate triangles, when it can estimate the correct scale (see e.g. the eagle dataset).

The algorithm outputs exactly the Delaunay triangulation of the input points restricted to the union of the D_i^r disks (Section 3.4). In the experiments, we observed that the behavior of our algorithm is very similar to the Cocone algorithm (Table 1). because the disks that we use do not deviate much from the co-cones. From a theoretical point of view, it would be interesting to characterize the configurations where the outputs of both algorithms match exactly. This probably involves some variants of ϵ-sampling conditions.

Regarding timings, our algorithm is the fastest, by two order of magnitudes, in some cases. However, one needs to take into account that the SuperCocone's implementation that we obtained from its authors is sequential, and could probably be easily parallelized. We think that our algorithm will still be faster since it does not need to compute any 3D Delaunay triangulation (it directly computes the Restricted 2D Delaunay triangulation).

Unlike Poisson reconstruction and its variants, our algorithm does not need coherently oriented normals, it solely uses normal directions. This can be interesting when such information is not available or when the input point set has a shape that is too complicated for greedy orientation algorithms, such as in Fig. 19. Our algorithm requires a moderate smoothing when the data presents alignment errors, but it does not require a per-sample resolution estimate as in [4].

Observing the relative timings of the different steps of our algorithm, manifold extraction and post-processing clearly dominate the core of the algorithm. However, they are needed by most geometry processing applications that rely on a clean manifold mesh. For future work, there is room for improvement in these steps. Our implementation of hole filling is not well optimized, not parallel, and we think it could probably be made two or three times faster.

The complete implementation in C++ is available in our open source GEOGRAM library [56].

Acknowlagement. B. Levy's work was partially funded by INRIA Exploragram grant and ANR MAGA and ROOT grants.

[1] T. K. Dey, R. Dyer, L. Wang, Localized cocone surface reconstruction, Computers and Graphics 35 (3) (2011) 483–491.
[2] J.-D. Boissonnat, A. Ghosh, Manifold reconstruction using tangential Delaunay complexes, Discrete and Computational Geometry 51 (1) (2014) 221–267.
[3] M. Kazhdan, H. Hoppe, Screened poisson surface reconstruction, ACM Trans. Graph. 32 (3) (2013) 29:1–29:13.
[4] S. Fuhrmann, M. Goesele, Floating scale surface reconstruction, ACM Trans. Graph. 33 (4) (2014) 46:1–46:11.
[5] J. R. Munkres, Elements of algebraic topology, Addison-Wesley, 1984.
[6] M. Berger, J. A. Levine, L. G. Nonato, G. Taubin, C. T. Silva, A benchmark for surface reconstruction, ACM Trans. Graph. 32 (2) (2013) 20:1–20:17.
[7] M. Berger, A. Tagliasacchi, L. Seversky, P. Alliez, J. Levine, A. Sharf, C. Silva, State of the Art in Surface Reconstruction from Point Clouds, in: Eurographics, 2014, pp. 161–185.
[8] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle, Surface reconstruction from unorganized points, SIGGRAPH 26 (2) (1992) 71–78.

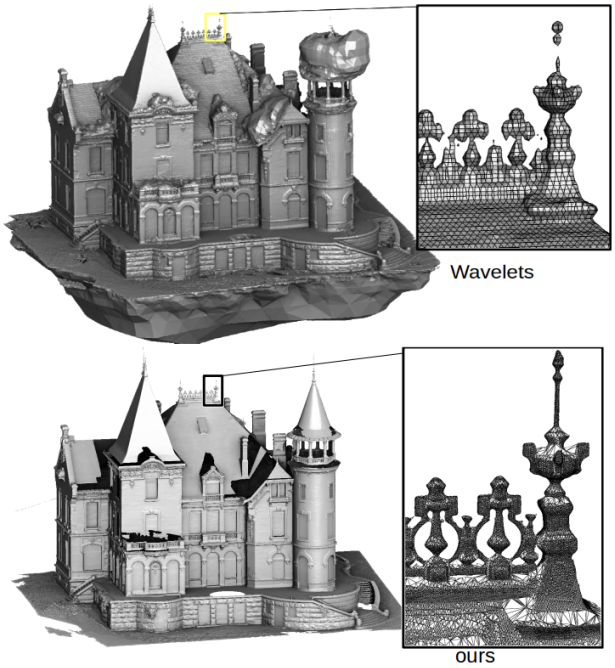


Figure 20: A raw point set with 33M points. On the top, the Wavelet algorithm builds, with depth=10, a coarse approximation mesh with 6M vertices. On the bottom, our algorithm builds a detailed mesh with 65M vertices. The total computation takes 360 seconds - 49s for 2 iterations of smoothing and normal estimation, 78s reconstruction, 61s surface extraction, 114s post-processing and 58s loading and saving.

[9] J.-D. Boissonnat, F. Cazals, Smooth surface reconstruction via natural neighbour interpolation of distance functions, Computational Geometry 22 (1 - 3) (2002) 185 – 203.

[10] B. Curless, M. Levoy, A volumetric method for building complex models from range images, in: SIGGRAPH, 1996, pp. 303–312.

[11] M. Kazhdan, M. Bolitho, H. Hoppe, Poisson surface reconstruction, in: Proceedings of the 4th SGP, 2006, pp. 61–70.

[12] J. Manson, G. Petrova, S. Schaefer, Streaming surface reconstruction using wavelets, in: SGP, 2008, pp. 1411–1420.

[13] P. Alliez, D. Cohen-Steiner, Y. Tong, M. Desbrun, Voronoi-based variational reconstruction of unoriented point sets, in: Proc. of the 5th SGP, 2007, pp. 39–48.

[14] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, C. T. Silva, Computing and rendering point set surfaces, IEEE Trans. on Vis. and Comp. Graphics 9 (1) (2003) 3–15.

[15] Y. Ohtake, A. Belyaev, M. Alexa, Sparse low-degree implicit surfaces with applications to high quality rendering, feature extraction, and smoothing, in: SGP, 2005, pp. 1–10.

[16] Y. Nagai, Y. Ohtake, H. Suzuki, Smoothing of partition of unity implicit surfaces for noise robust surface reconstruction, in: SGP, 2009, pp. 1339–1348.

[17] W. E. Lorensen, H. E. Cline, Marching cubes: A high resolution 3d surface construction algorithm, SIGGRAPH Comput. Graph. 21 (4) (1987) 163–169.

[18] N. J. Mitra, A. Nguyen, Estimating surface normals in noisy point cloud data, in: Proc. of the SCG, 2003, pp. 322–328.

[19] T. K. Dey, G. Li, J. Sun, Normal estimation for point clouds: A comparison study for a voronoi based method, in: Proc. of the Conference on Point-Based Graphics, 2005, pp. 39–46.

[20] H. Edelsbrunner, N. R. Shah, Triangulating topological spaces, in: Proc. of the 10th Annual Symposium on Computational Geometry, ACM, NY, USA, 1994, pp. 285–292.

[21] N. Amenta, M. Bern, Surface reconstruction by voronoi filtering, in: Proc. of the 14th Annual Symposium on Computational Geometry, ACM, NY, USA, 1998, pp. 39–48.

[22] N. Amenta, S. Choi, T. K. Dey, N. Leekha, A simple algorithm for homeomorphic surface reconstruction, in: Proc. of the 16th SCG, 2000, pp. 213–222.

[23] N. Amenta, S. Choi, R. K. Kolluri, The power crust, in: Proc. of the 6th SMA, 2001, pp. 249–266.

[24] F. Cazals, J. Giesen, Delaunay triangulation based surface reconstruction: Ideas and algorithms, in: Effective computational geometry for curves and surfaces, Springer, 2006, pp. 231–273.

[25] T. K. Dey, Curve and Surface Reconstruction: Algorithms with Mathematical Analysis, Cambridge Univ. Press, NY,USA, 2006.

[26] D. Cohen-Steiner, F. Da, A greedy delaunay-based surface reconstruction algorithm, The Visual Comp. 20 (1) (2004) 4–16.

[27] C.-C. Kuo, H.-T. Yau, A delaunay-based region-growing approach to surface reconstruction from unorganized points, Computer-Aided Design 37 (8) (2005) 825 – 835.

[28] R. Allegre, R. Chaine, S. Akkouche, A flexible framework for surface reconstruction from large point sets, Computers & Graphics 31 (2) (2007) 190 – 204.

[29] J.-D. Boissonnat, S. Oudot, Provably good sampling and meshing of surfaces, Graph. Models 67 (5) (2005) 405–451.

[30] H. Edelsbrunner, Discrete and Computational Geometry, Springer, Berlin, 2003, Ch. Surface Reconstruction by Wrapping Finite Sets in Space, pp. 379–404.

[31] E. A. Ramos, B. Sadri, Geometric and topological guarantees for the wrap reconstruction algorithm, in: Proc. of SODA '07, 2007, pp. 1086–1095.

[32] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, G. Taubin, The ball-pivoting algorithm for surface reconstruction, IEEE Trans. on Vis. and Comp. Graphics 5 (1999) 349–359.

[33] L. Di Angelo, P. Di Stefano, L. Giaccari, A new mesh-growing algorithm for fast surface reconstruction, Computer Aided Design 43 (6) (2011) 639–650.

[34] H. Edelsbrunner, E. P. Mücke, Three-dimensional alpha shapes, ACM Trans. Graph. 13 (1) (1994) 43–72.

[35] J. Digne, An Analysis and Implementation of a Parallel Ball Pivoting Algorithm, Image Proc. On Line 4 (2014) 149–168.

[36] J. Digne, J.-M. Morel, C.-M. Souzani, C. Lartigue, Scale space meshing of raw data point sets, Computer Graphics Forum 30 (6) (2011) 1630–1642.

[37] S. Funke, E. A. Ramos, Smooth-surface reconstruction in near-linear time, in: SODA '02, Society for Industrial and Applied Mathematics, PA, USA, 2002, pp. 781–790.

[38] D. Dumitriu, S. Funke, M. Kutz, N. Milosavljević, On the locality of extracting a 2-manifold in r3, in: 11th Scand. Workshop on Algorithm Theory, Vol. 5124, Springer, 2008, pp. 270–281.

[39] D. Dumitriu, S. Funke, M. Kutz, N. Milosavljević, How much geometry it takes to reconstruct a 2-manifold in r3, J. Exp. Algorithmics 14 (2010) 2:2.2–2:2.17.

[40] Y. J. Kil, N. Amenta, Gpu-assisted surface reconstruction on locally-uniform samples, in: IMR, 2008, pp. 369–385.

[41] D. Attali, J.-D. Boissonnat, A. Lieutier, Complexity of the delaunay triangulation of points on surfaces: the smooth case, in: Proc. of SGP, USA, 2003, pp. 201–210.

[42] M. Gopi, S. Krishnan, C. Silva, Surface reconstruction based on lower dimensional localized delaunay triangulation, Computer Graphics Forum 19 (3) (2000) 467–478.

[43] M. Gopi, S. Krishnan, A fast and efficient projection-based approach for surface reconstruction, in: High Performance Computer Graphics, Multimedia and Visu., 2002, pp. 1–12.

[44] R. Shewchuk, Star splaying: An algorithm for repairing delaunay triangulations and convex hulls, in: SCG, ACM, USA, 2005, pp. 237–246.

[45] H. Samet, Foundations of Multidimensional and Metric Data Structures, Morgan Kaufmann Publishers Inc., CA, USA, 2005.

[46] D. M. Mount, S. Arya, ANN: A library for approximate nearest neighbor searching, in: CGC Workshop on Computational Geometry, 1997, pp. 33–40.

[47] B. Levy, N. Bonneel, Variational Anisotropic Surface Meshing with Voronoi Parallel Linear Enumeration, in: Proc. of the 21st Int. Meshing Roundtable, 2012, pp. 33–40.

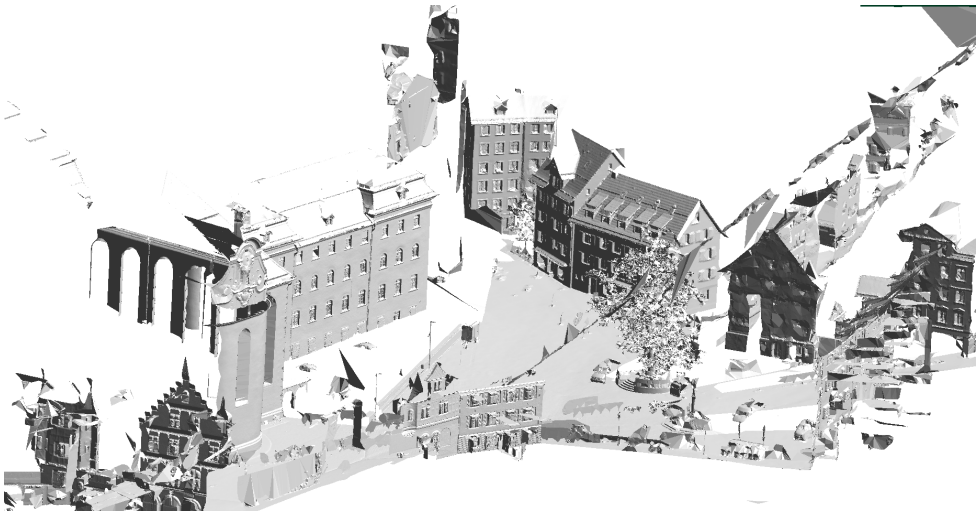


Figure 21: A raw point set of Saint Gallen place from [50]. The mesh has 14M vertices and has been built in 4 minutes.

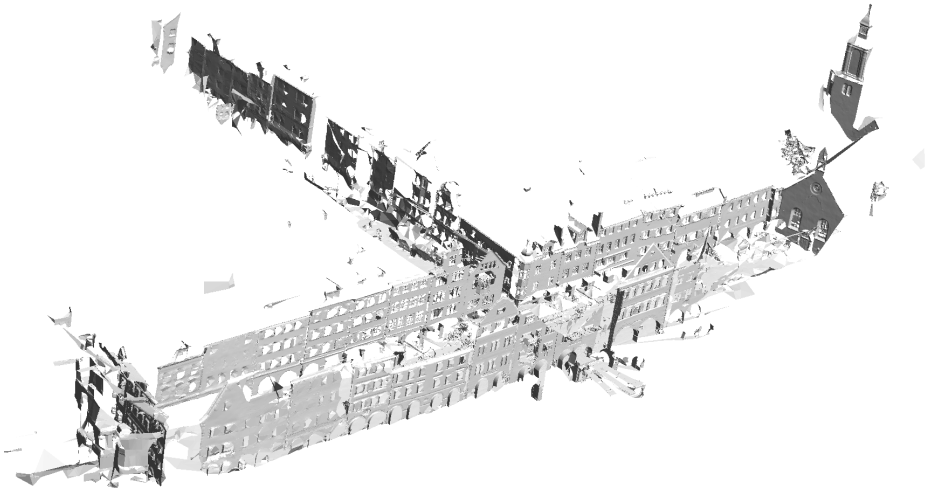


Figure 22: A raw point set of a market place from [50]. The mesh has 10M vertices and has been built in 3 minutes.

[48] P. Liepa, Filling holes in meshes, in: SGP, 2003, pp. 200–205.

[49] Smart multimedia, inc.
URL `http://www.smartmm.com`

[50] Large-scale point cloud classification benchmark.
URL `http://semantic3d.net`

[51] J. J. Digne, An Implementation and Parallelization of the Scale-Space Meshing Algorithm, Image Processing On Line 5.

[52] Aim shapes repository.
URL `http://visionair.ge.imati.cnr.it`

[53] Stanford computer graphics laboratory.
URL `http://graphics.stanford.edu/data/3Dscanrep`

[54] Farman institute 3d point sets.
URL `http://www.ipol.im/pub/art/2011/dalmm_ps`

[55] Statue model repository.
URL `http://lgg.epfl.ch/statues_dataset.php`

[56] Geogram, `http://alice.loria.fr/software/geogram`.